图文 43 秒杀系统的技术难点以及秒杀商品详情页系统的架构设

443 人次阅读 2019-12-02 07:00:00

详情 评论

秒杀系统的技术难点以及秒杀商品详情页系统的架构设计

石杉老哥重磅力作：《互联网java工程师面试突击》（第3季）**【强烈推荐】**：

全程真题驱动，精研Java面试中**6大专题的高频考点**，从面试官的角度剖析面试

（点击下方蓝字试听）

《互联网Java工程师面试突击》（第3季）

1、下一个要解决的问题是什么？

小猛在最近一段时间研究了MQ技术，并且合理的为公司搭建了一套RocketMQ集群之后，立马就将MQ技术引入到了订单系统里，解决掉了好几个之前比较棘手的问题

小猛看了一下问题清单：

~~（1）下单核心流程环节太多，性能较差~~
（2）订单退款的流程可能面临退款失败的风险
（3）关闭过期订单的时候，存在扫描大量订单数据的问题
~~（4）跟第三方系统耦合在一起，性能存在抖动的风险~~
~~（5）大数据团队要获取订单数据，存在不规范直接查询订单数据库的问题~~
（6）做秒杀活动时订单数据库压力过大

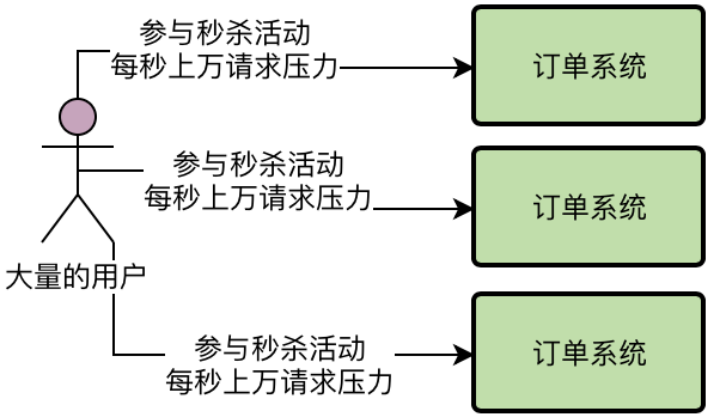
目前已经解决了3个技术问题了，还剩下的有订单退款失败、扫描大量订单、秒杀活动压力过大这3个问题了

这个时候小猛开始思索了，下一个应该解决哪个问题呢？正想着这个事儿呢，明哥来找他了。

明哥告诉了小猛一个消息，最近公司的运营花了很多钱做活动拉新用户，公司APP的日活用户一直在增长

现在已经明显发现每天高峰时间公司搞秒杀活动的时候，比以前有更多的用户在某个时间点蹲守在手机APP前。特价秒杀商品时间一到，就有大量的并发请求过来，系统压力非常大

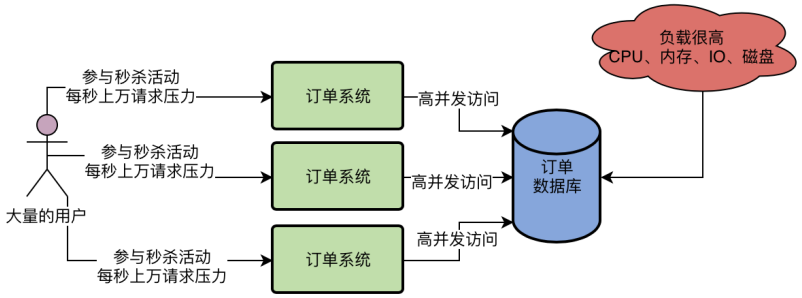
我们看下面的图：



如果仅仅是订单系统自己本身压力过大，还不是太大的问题。因为订单系统目前部署了20台4核8G的机器，整个集群抗每秒上万请求压力是可以的，即使后续用户量越来越大，大不了就是给订单系统加更多的机器就可以了。

但是这里有一个问题，20台订单系统的机器都是访问同一台机器上部署的MySQL数据库的，那一台数据库服务器目前经常在晚上秒杀活动的时候，瞬时并发量达到上万。

所以最近几天明显发现数据库的负载越来越高，比如CPU、IO、内存、磁盘的负载几乎都快要到极限了，看下面的图。



所以明哥说，整个公司各个技术团队都将要为秒杀活动进行系统优化，务必让各个系统可以用合理的架构、有限的机器资源去抗下来未来越来越多用户参与的秒杀活动，订单系统作为公司核心的交易系统，也必然要参与到本次优化中去。

而且这次架构优化，将会由各个技术团队的leader直接带队负责，小猛要全程参与到里面。

小猛一听，内心兴奋极了，因为终于有机会参与到公司的高并发系统架构优化里来了。

2、秒杀活动压力过大怎么办？难道是加机器吗？

今天明哥召开了一个订单技术团队内部的会议，给大家来介绍目前面临公司系统的整体情况，以及兄弟团队的秒杀架构优化方案。

首先明哥向订单团队的弟兄们抛出了**第一个问题**，秒杀活动目前压力过大，应该如何解决？是不是简单的堆机器或者加机器就可以解决的？

比如给订单系统部署更多的机器，是不是可以抗下更高的并发？

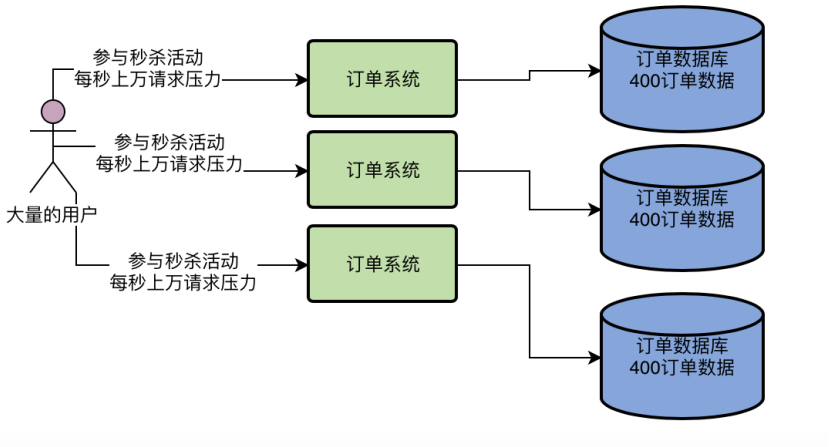
这个是没问题的，订单系统自己是可以通过部署更多的机器进行线性扩展的。

但是第二个问题来了，那么数据库呢？是不是也要部署更多的服务器，进行分库分表，然后让更多的数据库服务器来抗超高的数据库高并发访问？

这个思路是这样的，所谓分库分表，就是把目前的一台数据库服务器变成多台数据库服务器，然后把一张订单表变成多张订单表。

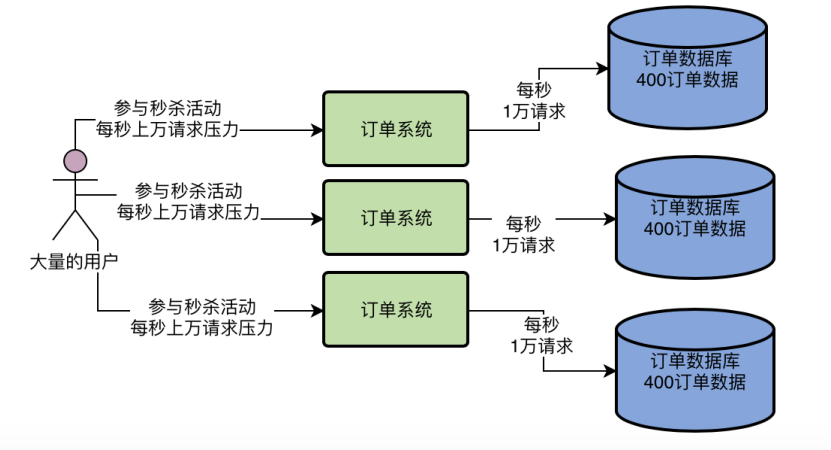
举个例子，目前假设订单表里有1200万条数据，然后有一台数据库服务器，如果我们现在变成3台数据库服务器，那么可以在每台数据库服务器里放400万订单数据，这就是所谓的分库分表

我们看下面的图



这种做法的好处是什么呢？

比如未来订单系统的整体访问压力达到了每秒3万请求了，此时订单系统通过扩容可以部署很多机器，然后其中1万请求写入到一台数据库服务器，1万请求写入到另一台数据库服务器，另外1万请求写入最后一台数据库服务器，就好像下面的图这样子。



这样不就可以通过增加更多的数据库服务器轻松的抗下更高的并发请求了吗？但是事实上这个方案大家觉得靠谱吗？

答案是**不太靠谱**的，除非是技术能力比较弱的公司，没有厉害的架构师去利用已有的技术合理设计优秀的架构，才会用这种堆机器的方法简单的来抗下超高的并发。

因为如果用堆机器的方法来解决这个问题，必然存在一个问题，就是随着你的用户量越来越大，你的并发请求越来越多，会导致你要不停的增加更多的机器

如果现在你每秒的并发请求量是1万，可能你就需要20台4核8G的订单服务器+1台高配置的数据库服务器，就可以扛下来了。

但是如果你未来用户量增长10倍，每秒有10万并发请求呢？难道你就直接让订单系统部署200台机器？然后将数据库服务器增加到10台？这样会导致你公司的服务器成本急剧飙升！

所以解决问题往往不能用这种简单粗暴堆机器的方案！

3、巧妙的架构设计帮公司节省巨大成本

会议进行到这里，明哥直接代表公司的技术高层做了一个总结：为了应对秒杀活动这种特殊场景，不能采取无限制的扩容服务器的方案，而应该是利用各种技术去合理设计更加优秀的架构，在有限的机器资源条件下，去抗下更高的并发！

因此我们不能仅仅依赖于最简单粗暴的堆机器的策略，而是要仔细的分析秒杀活动进行时的核心请求链路，然后精心设计架构，优雅的抗下越来越高的并发量。

4、不归订单管的部分：高并发的商品详情页请求

明哥接着介绍，其实秒杀活动主要涉及到的并发压力就是两块，**一个是高并发的读，一个是高并发的写。**

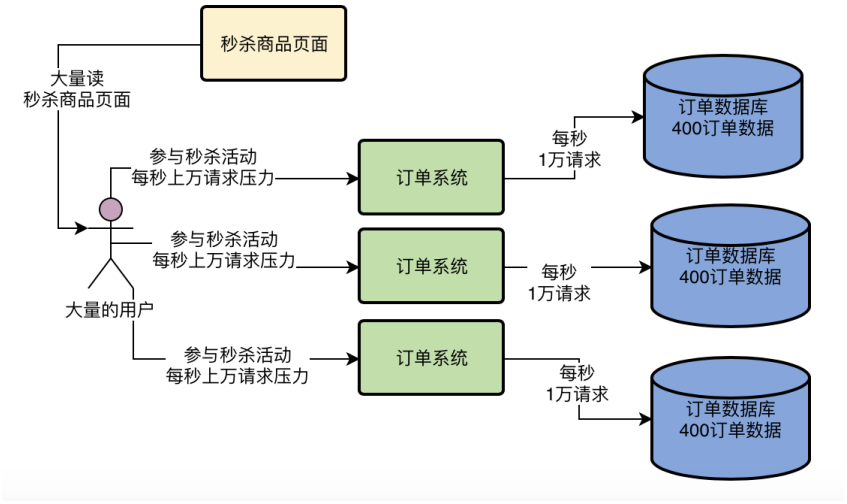
我们还是从用户参与秒杀活动的实际场景入手，一点一点从业务到技术的来进行分析。

首先大家可以思考一下，平时大量的用户是怎么参与到秒杀活动里来的?

往往是这样，很多人都知道我们的APP每天晚上比如8:30会有一波秒杀商品开始售卖，因此每次到了晚上8:30之前，就有很多用户会登录我们的APP，然后在APP前坐等秒杀特价商品。

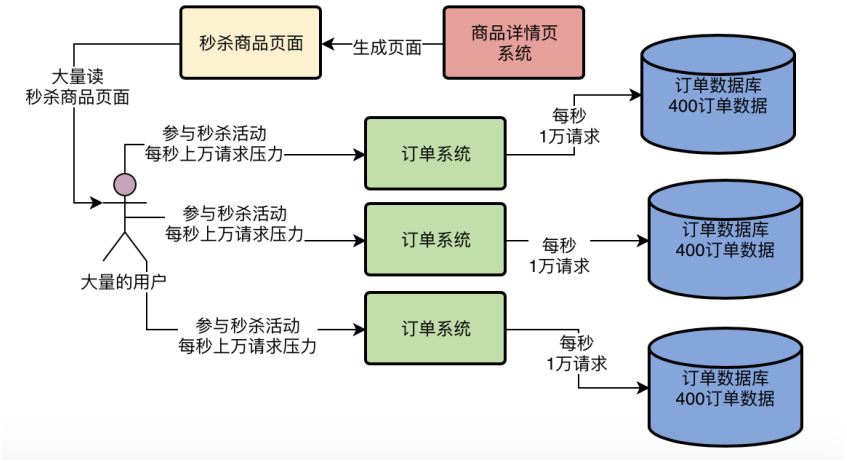
所以这个时候，必然出现一种场景，就是首先大量用户会拿着APP不停的刷新一个秒杀商品的页面

我们看下面的图



那么这些秒杀商品页面是从哪儿加载出来的呢?

本质上来说是从商品技术团队负责的商品详情页系统中加载出来的，我们看下面的图，图中引入了一个商品详情页系统的概念，他负责提供我们看到的各种秒杀商品页面。



所以首先这个商品详情页系统就是在秒杀活动开始之前最先被大量用户高并发访问的一个系统了！

大家可以思考一个问题，如果没有秒杀活动的时候，其实大量的用户是分散在不同的时间段里来逛我们的APP的，而且逛的是不同的人会看不同的商品的页面。

但是在秒杀活动的时候，他面临的第一个问题就是，可能几十万人，甚至百万级的用户，会同一时间频繁的访问**同一个秒杀商品的页面**

比如“3折抢购原价6888的手机，限售100台”这样的活动，可能有几十万人在8:30之前会集中访问这个秒杀商品的活动页面，对商品详情页系统造成过巨大的访问压力。

5、商品团队的秒杀架构优化：页面数据静态化

因此明哥接着开始讲解，商品技术团队是如何解决秒杀商品活动页面被同一个时间点的大量用户频繁访问，造成商品详情页系统压力过大的问题。

实际上商品技术团队针对这个问题，采取的是**页面数据静态化+多级缓存**的方案。

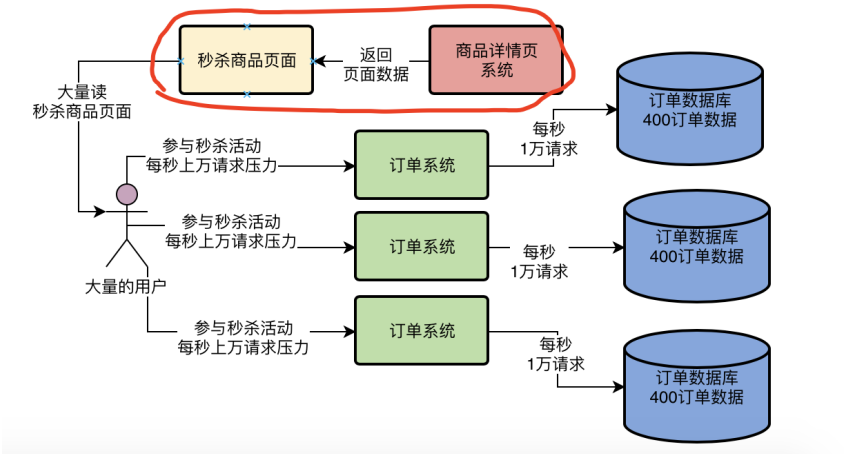
首先第一步，秒杀商品页面必须是将其数据做到静态化，这是什么意思呢？

简单来说是这样，如果让秒杀商品页面是动态化的，那么每次一个用户只要访问这个商品详情页，就必须发送一次请求到后端的商品详情页系统来获取数据。

比如商品的标题、副标题、价格、优惠策略、库存、大量的图片、商品详情说明、售后政策等等，这一大堆的东西都是商品详情页的数据。

那么你可以选择让用户浏览这个秒杀商品的时候，每次都发送请求到后台去加载这些数据过来，然后渲染出来给用户看这个商品页面，这就是所谓的动态模式。

我们看下面的图里画圈的地方，很明显就是这种方式。



如果这商品详情页里的大量数据都是存储在商品团队的数据库里的，那么岂不是大量的用户同时频繁访问这个商品详情页，会直接导致商品详情页系统承受高并发的访问？同时导致商品数据库承受高并发的访问？

所以首先需要将这个秒杀活动的商品详情页里的数据做成静态化的，也就是说提前就从数据库里把这个页面需要的数据都提取出来组装成一份静态数据放在别的地方，避免每次访问这个页面都要访问后端数据库。

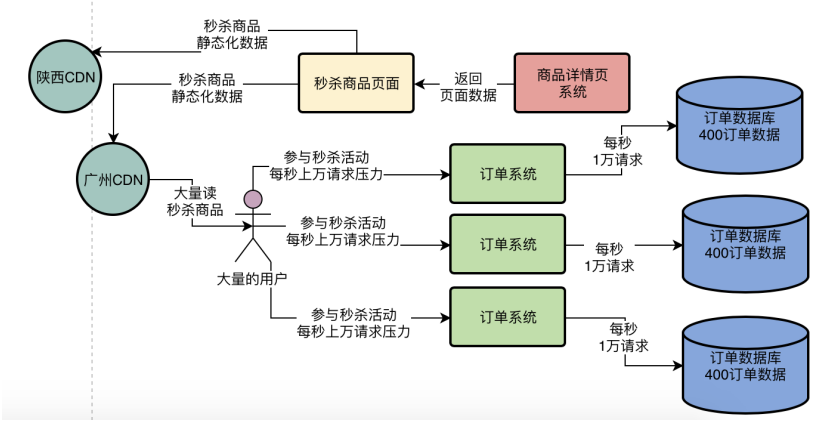
6、商品团队的秒杀架构优化：多级缓存

接着就是多级缓存的架构，我们会使用**CDN + Nginx + Redis**的多级缓存架构

什么意思呢？就是说秒杀商品详情页的数据，首先会放一份在离用户地理位置比较近的CDN上

CDN你大致可以这么理解。比如我们公司的机房在上海，系统也部署在上海，那么对于陕西的用户，难道每次都要发送请求到我们的上海机房里来获取数据吗？

不是，我们完全可以将一些静态化好的数据放在陕西的一个CDN上。同样对于广州的用户，可以把这些静态化好的数据放在广州的CDN上，这个CDN现在都是各种云厂商提供的服务，我们先看下面的图。



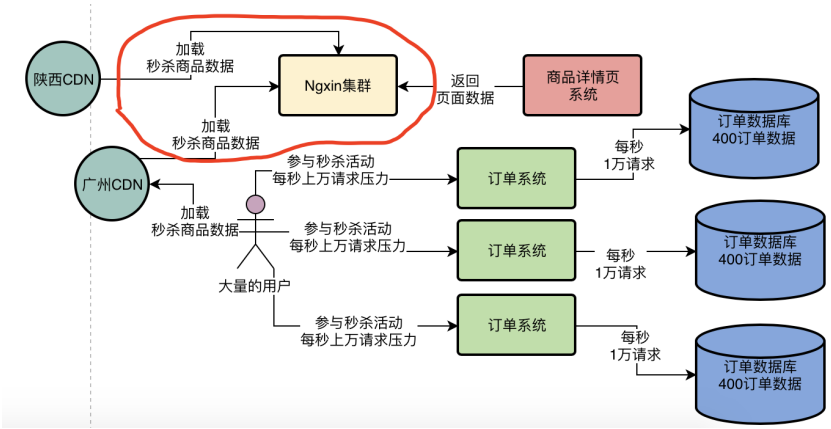
然后不同地方的用户在加载这个秒杀商品的详情页数据时，都是从就近的CDN上加载的，不需要每次请求都发送到我们公司在上海的机房去。

这个CDN缓存就是我们多级缓存架构里的第一级缓存。

那如果因为缓存过期之类的问题，CDN上没有用户要加载的商品详情页数据怎么办呢？

此时用户就会发送请求到我们公司的机房里的机器上去请求加载这个商品的数据了，这个时候我们需要在Nginx这样的服务器里做一级缓存。

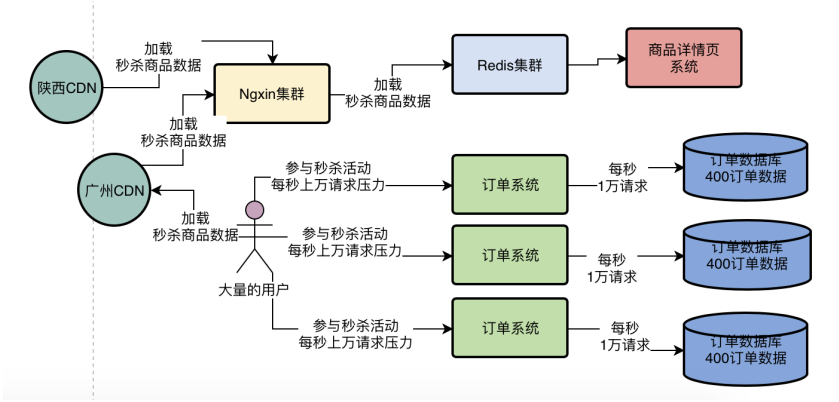
在Nginx中是可以基于Lua脚本实现本地缓存的，我们可以提前把秒杀商品详情页的数据放到Nginx中进行缓存，如果请求发送过来，可以从Nginx中直接加载缓存数据，不需要把请求转发到我们商品系统上去，看下面的图。



这个时候如果在Nginx服务器上也没加载到秒杀商品的数据呢？

比如同样因为Nginx上的缓存数据过期之类的问题，导致没找到我们需要的数据。

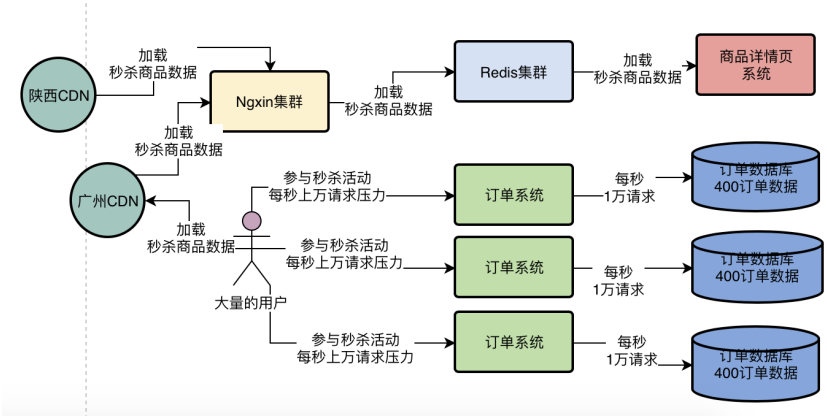
此时就可以由Nginx中的Lua脚本发送请求到Redis集群中去加载我们提前放进去的秒杀商品数据，如下面的图。



如果在Redis中还是没有找到呢?

那么就由Nginx中的Lua脚本直接把请求转发到商品详情页系统里去加载就可以了，此时就会直接从数据库中加载数据出来，如下图所示

但是一般来说数据一般是可以从CDN、Nginx、Redis中加载到的，可能只有极少的请求会直接访问到商品系统去从数据库里加载商品页数据。



通过这样的一套方案，我们就可以把用于秒杀活动的商品详情页数据进行静态化，然后把静态化以后的一串商品数据（比如可能就是一个大的JSON串）放到CDN、Ngxin、Redis组成的多级缓存里去，这样大量的用户同时访问这个秒杀商品页面就对我们的商品系统本身没什么压力了。

因为分布在全国各地的用户的大量请求都是分散发送给各个地方的CDN的，所以CDN就分摊掉了大量的请求。而即使请求到达了我们的后台系统，都是由轻松单机抗10万+并发的Nginx和Redis来返回商品数据的。

7、今天内容的一点小总结

今天我们借着明哥的视角给大家分析了一下秒杀场景下的堆机器方案的弊端，同时从秒杀活动发生的场景入手，分析了一下在秒杀活动发生的某个时间点前后，大量的用户会集中的去访问这个秒杀商品的页面。

因此为了针对这个问题进行优化，我们讲了商品技术团队需要做的数据静态化以及多级缓存的架构。

实际上秒杀系统是一个非常复杂的系统，里面涉及的细节是很多的，如果真的要从0开始带着大家讲清楚一个秒杀系统涉及到的方方面面和所有细节，至少需要一个专栏几十篇文章的内容才能说清楚。

因此在这个专栏里，我们主要是要借着秒杀场景去讲一下RocketMQ的限流削峰的功效，所以对秒杀系统本身的很多细节我们并没有涉及，主要是从整体角度讲一下秒杀系统的架构设计和思路，还望很多对秒杀系统的实现细节有兴趣的朋友理解。

End

专栏版权归公众号**狸猫技术窝**所有

未经许可不得传播，如有侵权将追究法律责任

狸猫技术窝其他精品专栏推荐：

《从零开始带你成为JVM实战高手》

《21天Java 面试突击训练营》（分布式篇）（现更名为：**互联网Java工程师面试突击第2季**）

互联网Java工程师面试突击（第1季）

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情在评论区留言提问，我会逐一答疑

如何加群：购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群**

具体加群方式，请参见目录菜单下的文档：《付费用户如何加群？》**（购买后可见）**

Copyright © 2015-2019 深圳小鹅网络技术有限公司 All Rights Reserved. 粤ICP备15020529号

小鹅通提供技术支持